**Kubernetes**

**What Is Kubernetes**

Kubernetes is a container management technology developed in Google lab to manage containerized applications in different kind of environments such as physical, virtual, and cloud infrastructure. It is an open source system which helps in creating and managing containerization of application.

Kubernetes is an open-source container management (orchestration) tool,Which automates container deployment, scaling & descaling of containers & container load balancing?

Features:

### Automatic Binpacking

Kubernetes automatically packages your application and schedules the containers based on their requirements and available resources while not sacrificing availability. To ensure complete utilization and save unused resources, Kubernetes balances between critical and best-effort workloads.

### Self-Healing

Personally, this is my favorite feature. Kubernetes can automatically restart containers that fail during execution and kills those containers that don’t respond to user-defined health checks. But if nodes itself die, then it replaces and reschedules those failed containers on other available nodes.

### Horizontal Scaling

Kubernetes needs only 1 command to scale up the containers, or to scale them down when using the CLI. Else, scaling can also be done via the Dashboard (kubernetes UI).

**Automated Rollback**

Sometimes you may want to rollback a Deployment; for example, when the Deployment is not stable, such as crash looping. By default, all of the Deployments rollout history is kept in the system so that you can rollback anytime you want.

Kubernetes progressively rolls out changes and updates to your application or its configuration, by ensuring that not all instances are worked at the same instance. Even if something goes wrong, Kubernetes will rollback the change for you.

### Secret & Configuration Management

Kubernetes can help you deploy and update secrets and application configuration without having to rebuilding your entire image and without having to exposing your secrets in your stack configuration or anything.

**Health Check of nodes and Containers**  
Kubernetes also helps to check constantly the health of nodes and containers.

**Compare Docker and Kubernetes:**

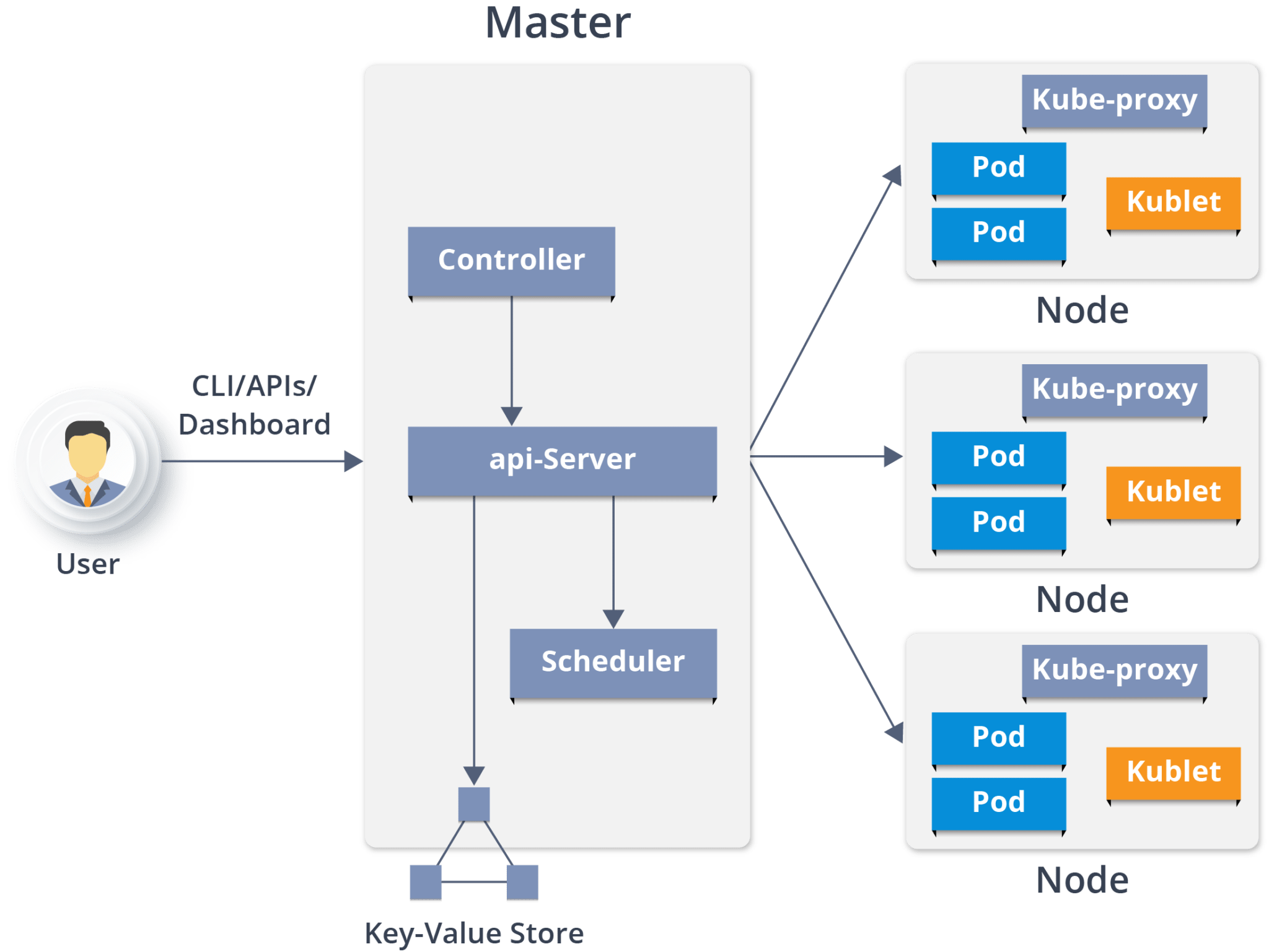
**Docker:** docker is a containerization platform.

### Kubernetes: is container management platform, which means that once we have containerized of application with help of docker containers or linux containers and when scaling of those containers to a big number like 50 to 100, this situation comes in kubernetes, so when we multiple containers need to managed kubernetes helps.

### How is Kubernetes different from Docker Swarm?

|  |  |  |
| --- | --- | --- |
| **Features** | **Kubernetes** | **Docker Swarm** |
| **Installation & Cluster Config** | Setup is very complicated, but once installed cluster is robust. | Installation is very simple, but the cluster is not robust. |
| **GUI** | GUI is the Kubernetes Dashboard. | There is no GUI. |
| **Scalability** | Scaling up is slow compared to swarm  but guarantee strong cluster sate | Scaling up faster than Kubernetes, but cluster  Strength not as robust. |
| **Auto-scaling** | Kubernetes can do auto-scaling. | Docker swarm cannot do auto-scaling. |
| **Load Balancing** | Loan balancing Manual confinguration required for load balancing traffic between different containers and pods. | Docker swarm does auto load balancing of traffic between containers in the cluster. |
| **Rolling Updates & Rollbacks** | Can deploy rolling updates and does automatic rollbacks. | Can deploy rolling updates, but not automatic rollback. |
| **DATA Volumes** | Can share storage volumes only with the other containers in the same pod.  Pod is nothing but group related containers  , logical group of containers together | Can share storage volumes with any other container. |
| **Logging & Monitoring** | In-built tools for logging and monitoring. | 3rd party tools like ELK stack should be used for logging and monitoring. |

## ****Kubernetes Architecture/Kubernetes Components****



Kubernetes Architecture has the following main components:

* Master nodes
* Worker/Slave nodes
* Distributed key-value store(etcd.)

## Master Node

It is the entry point for all administrative tasks which is responsible for managing the Kubernetes cluster. There can be more than one master node in the cluster to check for fault tolerance. More than one master node puts the system in a High Availability mode, in which one of them will be the main node which we perform all the tasks.

For managing the cluster state, it uses**etcd** in which all the master nodes connect to it.

Let us discuss the components of a master node. As you can see in the diagram it consists of 4 components:

**API server:**

* Performs all the administrative tasks through the API server within the master node.
* In this REST commands are sent to the API server which validates and processes the requests.
* After requesting, the resulting state of the cluster is stored in the distributed key-value store.

**Scheduler:**

* The scheduler schedules the tasks to slave nodes. It stores the resource usage information for each slave node.
* It schedules the work in the form of Pods and Services.
* Before scheduling the task, the scheduler also takes into account the quality of the service requirements, data locality, affinity, anti-affinity, etc.

**Controller manager:**

* Also known as **controllers**.
* It is a daemon which regulates the Kubernetes cluster which manages the different non-terminating control loops.
* It also performs lifecycle functions such as namespace creation and lifecycle, event garbage collection, terminated-pod garbage collection, cascading-deletion garbage collection, node garbage collection, etc.
* Basically, a controller watches the desired state of the objects it manages and watches their current state through the API server. If the current state of the objects it manages does not meet the desired state, then the control loop takes corrective steps to make sure that the current state is the same as the desired state.

**What is the ETCD?**

* etcd is a distributed key-value store which stores the cluster state.
* It can be part of the Kubernetes Master, or, it can be configured externally.
* etcd is written in the Go programming language. In Kubernetes, besides storing the cluster state (based on the **Raft Consensus Algorithm**) it is also used to store configuration details such as subnets, ConfigMaps, Secrets, etc.
* A raft is a consensus algorithm designed as an alternative to Paxos. The Consensus problem involves multiple servers agreeing on values; a common problem that arises in the context of replicated state machines. Raft defines three different roles (Leader, Follower, and Candidate) and achieves consensus via an elected leader

Now you have understood the functioning of Master node. Let’s see what is the Worker/Minions node and its components.

## Worker Node (formerly minions)

It is a physical server or you can say a VM which runs the applications using Pods (**a pod scheduling unit**) which is controlled by the master node. On a physical server (worker/slave node), pods are scheduled. For accessing the applications from the external world, we connect to nodes.

Let’s see what are the following components:

**Container runtime:**

* To run and manage a container’s lifecycle, we need a container runtime on the worker node.
* Sometimes, Docker is also referred to as a container runtime, but to be precise, Docker is a platform which uses containers as a container runtime.

**Kubelet:**

* It is an agent which communicates with the Master node and executes on nodes or the worker nodes. It gets the Pod specifications through the API server and executes the containers associated with the Pod and ensures that the containers described in those Pod are running and healthy.

**Kube-proxy:**

* Kube-proxy runs on each node to deal with individual host sub-netting and ensure that the services are available to external parties.
* It serves as a network proxy and a load balancer for a service on a single worker node and manages the network routing for TCP and UDP packets.
* It is the network proxy which runs on each worker node and listens to the API server for each Service endpoint creation/deletion.
* For each Service endpoint, kube-proxy sets up the routes so that it can reach to it.

### Pods

A pod is one or more containers that logically go together. Pods run on nodes. Pods run together as a logical unit. So they have the same shared content. They all share the same IP address but can reach other Pods via localhost, as well as shared storage. Pods don’t need to all run on the same machine as containers can span more than one machine. One node can run multiple pods.